DOCUMENT RESUME

ED 292 813 TM 011 125

AUTHOR Foshay, Wellesley R.
TITLE What We Know (and What We Don't Know) about Training of Cognitive Strategies for Technical Problem Solving.
PUB DATE Apr 87
NOTE 18p.; An earlier version of this paper was presented at the Annual Meeting of the American Educational Research Association (Washington, DC, April 20-24, 1987).
PUB TYPE Reports - Research/Technical (143) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS Algorithms; *Cognitive Processes; Cognitive Psychology; Heuristics; Instructional Effectiveness; Learning Theories; *Problem Solving; Schemata (Cognition); *Task Analysis
IDENTIFIERS *Troubleshooting

ABSTRACT

The topic of teaching troubleshooting is examined as an example of the teaching of cognitive strategies for technical problem solving. The traditional behavioral approach to teaching troubleshooting has essentially been algorithmic. Recent cognitive research suggests an approach founded first on task analysis and characterized by: (1) analysis of the symbol system used; (2) representation of the system under study; (3) identification of failure modes; (4) specific solution algorithms; and (5) derived heuristics which may be applied to guide linking of the detailed solution algorithms. Instructional strategies then would be developed for each of these areas. The teaching of troubleshooting for IBM's CICS application programming system is given as an example of the practical application of these principles. Much remains to be confirmed about teaching troubleshooting. Research on the learning psychology side of the cognitive field has surpassed research on the instructional psychology side. An instructional psychology should be developed using cognitive psychology as a basis. (SLD)

WHAT WE KNOW (AND WHAT WE DON'T KNOW) ABOUT TRAINING OF COGNITIVE STRATEGIES FOR TECHNICAL PROBLEM SOLVING

Wellesley R. Foshay
Advanced Systems, Inc.

An earlier version of this paper was presented to the American Educational Research Association, Washington, D.C., April, 1987.

The helpful comments of professors R.M. Gagné and J.M. Scandura on an earlier draft of this paper are gratefully acknowledged.

Let us start by identifying a fundamental dilemma of any practicing instructional developer: the models by which we work have considerable practical utility, and they are generally well grounded in theory. But the business of translating theory into practice is always risky. As Cunningham (1986) points out,

> ...we must rid ourselves once and for all of the notion that science will produce "truth," fixed and immutable for all time. Our notion of generalizability must be radically altered. We must allow ourselves the discomfort that comes when we realize that we are less certain about things than we previously imagined.

In this view, theory in principle does not lead smoothly and unambiguously to generalizable prescriptions for practice. Instead, practitioners must use theory as a "point of view" or a means for forming expectations of how real world problems will behave. These expectations help the practitioner understand the complexity of real world problems, and they help structure decisions about what to do. While it is true that the map is not the territory, it is also true that to navigate in the territory, one must have a map: theories are only abstractions of reality, but reality can only be understood through the point of view provided by theory.

And so it is now, as instructional developers seek to incorporate the theory of information processing cognitive psychology into their practical models of instructional design. As always, the research is incomplete and in some cases contradictory. But even so, we must ask if there are models, modes of analysis, or prescriptive principles which are powerful enough to improve the practice of an instructional developer who uses them.

The answer is a (heavily qualified) "yes," in the opinion of recent reviews, such as Fredericksen (1984) and André (1986). To evaluate these opinions, it is appropriate to examine how thinking about instructional design might change to incorporate cognitive theory. This examination also will help illuminate some unanswered theoretical questions and identify some tools which practitioners will need in order to effectively modify their design practice.

Let us take as the point of departure the topic of teaching trouble shooting, as one might find it in content areas such as electronic circuit fault detection, mechanical system repair, computer software debugging, or medical diagnosis. Troubleshooting has the advantage of being a common subject of training courses, and also a common topic of cognitive research. However, troubleshooting is generally well structured compared to other kinds of problem solving (such as design tasks), so the degree of generalizability to other cognitive tasks must be questioned.

First, we will quickly review common (behaviorally based) design practices for training in troubleshooting. Then, we will examine some of the cognitively based recommendations for teaching problem solving which have been recently published, and see how they might be applied to design of troubleshooting training. Finally, we will identify some unanswered questions of importance to practitioners seeking to incorporate cognitive principles into their instructional designs. As part of the discussion, we will note certain key divergences between Scandura's Structural Learning Theory (SLT) and other cognitive theories.

BEHAVIORALLY BASED DESIGN PRACTICES FOR TROUBLESHOOTING TRAINING

The behaviorally based approach to teaching troubleshooting is essentially algorithmic. For example, popular treatments such as Mager's (1982) often include recommendations such as these:

1. Identify the system's most common faults.
2. Derive one or more algorithm(s) for troubleshooting each common fault, using a split half strategy. A full analysis includes identification of conditions, actions, and feedback for each step.
3. Sequence instruction in each algorithm (or algorithm segment) using sequencing rules (e.g., teach prerequisite parts before wholes).
4. Teach each algorithm (or algorithm segment) separately, teaching the steps in retrograde sequence. Structure practice of each step so it includes:
   - realistic stimuli (conditions)

- o realistic responses
- o immediate feedback on accuracy of the response, in detail

Continue practice until the behavior is satisfactorily shaped.

Limitations of the Behavioral Approach. Instructional design strategies incorporating these features have been in use for over twenty years, and they have been shown repeatedly to be effective. However, the behavioral approach has not been without critics. For example, Duncan (1985) makes these points:

1. Detailed procedure analysis of the sort required is costly. Each system fault requires a separate algorithm, or algorithm segment.
2. Technicians resist using a fully algorithmic approach.
3. The algorithms are very situation specific, and thus expensive to update.
4. Retention of algorithm details is a constant problem, because most of the faults (and their associated algorithms) are rarely encountered— and thus rarely practiced.
5. Transfer of troubleshooting skills to new systems or new faults is relatively low.

It may be that experience with these limitations are at the root of the common practice of using job aids to record troubleshooting algorithms, wherever possible. When they can be used in the work situation, job aids effectively offset the problem of retention, and the need for transfer is greatly reduced. However, the cost of the analysis and the need for constant updating still remains.

Recent cognitive research suggests some ways of overcoming some of these limitations. To see how, let us next summarize some key findings from that research.

KEY COGNITIVE PRINCIPLES OF RELEVANCE TO TROUBLESHOOTING

Of the various lines of cognitive research, two of perhaps greatest interest to instructional designers teaching problem solving deal with knowledge representation and with the things that expert problem solvers know. While these findings are familiar, we will review each to facili-

tate later discussion.

Knowledge Representation. In general, cognitive researchers have tended to draw distinctions between different types of knowledge. For example, André's chapter on problem solving in a recent introductory text (Phye and André, 1986) distinguishes between:

- Concepts, or schemata, which are stored as sets of multiple discrimination rules and as prototype examples; and
- Production systems, including rules, principles and skills, which specify the conditional relationships between concepts.

In addition, André makes the familiar distinction between:

- Declarative knowledge, which supports the ability to classify or define; and
- Procedural knowledge, which supports the the ability to per form.

An exception has been Scandura's Structural Learning Theory (Scandura, 1986). In SLT, both declarative and procedural knowledge are subsumed in a single rule structure. In the rule structure, higher order rules subsume lower order ones, and separate knowledge structures are not postulated.

Expert Knowledge. In studying differences between expert and novice problem solvers, André's review distinguishes two types of knowledge important for our purposes:

- Heuristic knowledge. Heuristics, or generally applicable (but imprecise) production systems, are used by experts to control problem representation and selection of solution strategy.
- Domain Specific knowledge. Experts have mastered large arrays of knowledge specific to a particular domain. This knowledge probably includes:

  the symbol system in use
  the structure of the system
  types of problems
  problem solution algorithms
  strategies for applying heuristic knowledge to domain specific knowledge.

Again, there is a contrast with SLT. SLT does not allow for extra domain knowledge, other than a single "goal switching" control mechanism, which is taken to be innate. What others call heuristics are taken within SLT to be higher order rules, which may be applied to generate lower order (content specific) rules as required. Another important phenomenon is automaticity, or the ability to process certain kinds of highly standardized algorithms rapidly and with minimal cognitive processing load (Schneider & Shiffrin, 1977). There is growing evidence that automaticity is important because it frees up attentional power for other, less standardized tasks. Elio (1986) suggests that this effect applies to production systems, but not to concepts. In other words, rapid recall of facts does not necessarily facilitate learning of advanced skills, but rapid performance of component subskills does.

Finally, the expert's reasoning process itself has been studied. Elstein, Schulman and Sprafka (1978) characterized physician's reasoning process as hypothetico-deductive, meaning that expert physicians usually formulated a small number of hypotheses early in the problem solving process, then gathered the information with the greatest power in discriminating between competing hypotheses. This is essentially a refinement of earlier views (for example, Cagné, 1954) that activities such as fault identification are essentially tree structured deductive tasks involving sequential discrimination of multiple cues. However, more recent work (Patel & Groen, 1986) suggests that this kind of backward reasoning may not be used universally. At least in the case of problems of moderate difficulty, a forward reasoning process seems to be more characteristic of experts.

With these basic concepts in mind, let us now turn to the problem of teaching troubleshooting by addressing it as a subset of the general issue of teaching problem solving.

HOW TO TEACH TROUBLESHOOTING

If we approach the task of designing instruction for troubleshooting by using a standard instructional design model, it makes sense to

first discuss the analysis of the task, and then to address the selection of appropriate instructional strategies.

Analyzing the Task. The theory reviewed above leads to some significant departures from conventional task/content analysis. The analysis procedure might look like this:

1. If experts use a symbol system to describe the system, identify it. Then analyze the characteristics of the symbol system through a conventional concept analysis.

This recommendation is based on the finding that knowledge of an appropriate symbol system is the first level of an expert's domain specific knowledge.

2. Using this symbol system, represent the system under study, identifying components within the system. The level of detail of analysis should be such that the smallest unit of analysis corresponds to the smallest component to be acted upon in troubleshooting. The analysis should include at least the name and function of each system component.

This corresponds to the identification of concepts or schemata for an expert's domain specific knowledge, or the lowest order of rules in an SLT rule structure.

3. For each component, identify the failure modes and the probabilities and costs associated with each failure. Then group the failure modes into categories.

This corresponds to identification of the expert's knowledge of the types of problems which occur. The recommendation to attach probabilities and costs to the failures is consistent with Elstein, Schulman and Sprafka (1978), and also with work done with some expert systems.

4. For each category of failure, identify the solution algorithm

for isolating the problem. The algorithms are specific indicators or tests which isolate compnent failures. They are not complete algorithms for troubleshooting the entire system. It may also be worthwhile to identify the relative cost and reliability of each test.

This corresponds to an expert's knowledge of problem solution algorithms for each problem type, or higher order rules within SLT. There is some evidence from studies such as Elstein, Schulman and Sprafka (1978) that experts also take into account the cost and reliability of each test when weighing the information value of each alternative.

5. For the system as a whole, derive heuristics which may be applied to guide linking of the detailed solution algorithms to find a particular fault.

This corresponds to an expert's knowledge of problem solving strategies, or the highest order rules within SLT.

Compared to conventional behavioral analysis, this task analysis is much more comprehensive, yet potentially simpler to perform. It shows the structure of the system when it works, and when it doesn't work; it also shows the production systems used by an expert when troubleshooting at a precise and a heuristic level. However, it does not do what a behavioral analysis would: it does not map the exact algorithms needed to trouble shoot every particular fault in a specific system. Instead, it assumes that detailed troubleshooting algorithms are constructed by experts as they are confronted with each malfunction. To do this, the experts draw on all five types of knowledge analyzed above. If all five types of knowledge are taught to learners in an effective way, the assumption is that they also will be able to construct the algorithms as experts do.

This approach to cognitive task analysis also differs from that proposed by Brien and Duchastel (1986). In their technique, discrete learner objectives are developed for reproduction competencies (for which the learner is expected to recall the algorithm), and production competencies (which the learner must generate). The production competencies corre-

spond to the troubleshooting algorithms mentioned in point 4 above. However, Brien and Duchastel do not seem to make separate provisions for heuristic knowledge, as recommended in point 5 above. Furthermore, structural analysis of the relationships between the rules in the knowledge structure is not discussed.

With the task analysis complete, let us turn to the design of the instructional strategy.

Instructional Strategies. It would seem logical to teach each of the five types of knowledge with a separate instructional strategy. However, there is considerable controversy over whether this is really effective. For example, Duncan (1985) reported improvements in fault finding efficiency, but not accuracy, when various problem representation strategies were taught. Improvements in accuracy came only with practice. An attempt to directly represent production systems to physics students by Hewson and Posner (1984) yielded mixed results. On the other hand, Boekaerts (1985) has concluded that greater transfer occurs when concepts and production systems are taught in an integrated fashion as a complete system representation. Scandura (1986) concurs from both theoretical and empirical evidence that use of separate instructional strategies is both less efficient and less effective.

While integrated teaching of concepts and production systems may be best for declarative knowledge, the opposite may be true of procedural knowledge such as the specific algorithms and general heuristics known by experts. Chaiklin (1984) has argued that verbal representation of procedural rules assists novices as they master the rules, even though the verbal representations drop out when experts use the procedures. Furthermore, Chaiklin argues that even experts may return to verbal representation of the rules when confronted by a difficult problem.

These recommendations are far from certain. Each of them could be challenged by citing conflicting research findings. However, it appears (at least to this author) that they represent the most persuasive positions among current work. Combining them with more widely recognized instructional strategy recommendations leads to this instructional strategy for teaching troubleshooting:

1. First, teach the symbol system used to represent the problem. The symbol system is essentially a system of declarative knowledge involving both terms and concepts. Procedures for teaching terms and concepts using sequences of definitions, positive and negative examples have been well documented, most recently by Tennyson and Cocchiarella (1986).

2. Then, teach the system under study, including both its components and their causal relationships (how the system works). This corresponds to the recommendation to teach concepts and production systems simultaneously. Procedures for teaching in this way are outlined by Tennyson and Cocchiarella (1986).

3. Continue teaching the system by identifying the failure modes of each type of system component and how each failure affects the system as a whole (how it works when it doesn't work). Again, this is an example of teaching both concepts and production systems at the same time, so the instructional procedures would be as in the previous step. However, in this case the content directly concerns system malfunctions.

If the designer wishes to teach probabilities and costs for each class of failure, these would be two kinds of concept attribute to teach.

4. Teach algorithms for isolating each type of component failure. These would be specific algorithms which provide definitive tests to isolate each class of component failure. They would be taught using standard procedure teaching strategies, such as those outlined by Salisbury, Richards and Klein (1985).

5. Teach heuristics for troubleshooting the system in general. Use a practice strategy which asks the learner to state the heuristics and use them by applying them to troubleshooting the system. In other words, practice of the heuristics would involve asking the learner to verbalize them as they generate and state specific algorithms for troubleshooting given problems in specific systems. This is an application of Chaiklin's (1984) recommendation and is also consistent with Gagné (1954).

A general consideration is whether, in each step, the learner should continue practice until automaticity is achieved. As previously noted, Elio's (1986) argument that automaticity facilitates learning of production systems but not concepts suggests that extended practice should be planned at least for steps 2 through 4, especially in exercises requir-

ing the learner to apply the production systems by stating or predicting functional relations. Reasoning from Duncan's (1985) review, one would expect the extended practice to affect only accuracy and speed of solving such exercises.

It should also be noted that steps 2 through 5 would involve heavy use of simulation to teach the production systems and heuristics. The realism of such simulations should be carefully regulated. The simulations should provide only the practice of relevance for each step. Consequently, they are likely to be relatively unrealistic, allowing for considerable intrusion of instruction into the simulation. This is consistent with Munro, Fehling and Towne's (1985) recommendation for computer based simulations.

The strategy described above represents a plausible application of some current research. However, many of the recommendations may be open to challenge based on studies other than those cited. Furthermore, the author could identify no examples of actual instructional systems using all five of the recommended strategies. To illustrate some of the recommendations, the next section will briefly describe a commercial training product which is applies some of them. Then, a final section will summarize some of the points of controversy surrounding the instructional strategy proposed above.

CICS MAINTENANCE: A PRACTICAL APPLICATION OF SELECTED PRINCIPLES

The CICS Maintenance product (Robbins and Connors, 1987) was developed by Advanced Systems,Inc. as the introductory course in its curriculum to teach IBM's CICS application programming system. The course was implemented in a combination of linear video with a printed text exercise manual (Video Assisted Instruction, or VAI), and level III interactive video, with a supplementary printed text component and a pocket reference job aid (Interactive Video Instruction, or IVI). The target audience for the course is proficient applications programmers and systems analysts who are learning CICS for the first time. Needs analysis showed that beginning

CICS programmers are first assigned to maintenance tasks, such as fixing bugs or adding features to existing CICS programs. In spite of this, there are no reference manuals or training products which explicitly teach this skill. Instead, it is widely regarded by CICS experts as one which ard to learn and not usually mastered until the programmer has had a ..mber of years experience with CICS.

The course thus represents a radical departure from conventional CICS entry level training in at least two ways. First, it is an explicit attempt to teach novices a skill which experts widely believe "can be learned, but not taught." Second, the course is an explicit attempt to teach a high level problem solving skill.

The course is not a full implementation of the analysis and design strategy outlined above. However, some of the principles identified were applied. These are described below.

Task Analysis

- **Analysis of the symbol system** was in this case analysis of the CICS programming language itself, using conventional methods of concept analysis.

- **Representation of the system under study.** The system under study was CICS programs, as implemented in typical programming structures. Analysis thus included identification of typical programming structures as executed in blocks of CICS code. Analogies to COBOL programming also were identified.

- Failure modes. The failure modes included both commonly occurring bugs and frequent maintenance requests. These were identified by expert CICS programmers and then grouped into categories. No attempt was made to exhaustively identify every possible bug or maintenance requirement. Instead, only the problems which were most commonly occurring and most typical of their category were analyzed.

- Solution algorithm. Specific solution algorithms, such as debugging tests or modification procedures, were not separately identified in task analysis.

- Heuristics. For each category of bug or maintenance request, "rules of thumb" were identified, in the form of small productions for "things to check" or "points to consider." These then were sequenced into a flowchart (a format chosen because of its familiarity to the audience), even though the result was not fully deterministic.

Instructional Strategies.

- Teaching the symbol system was done early in the course, in linear video with text based practice, using standard concept teaching sequences involving careful isolation and presentation of concept attributes, use of positive and negative example sequences, and spaced practice involving generation of simple code and discrimination of positive and negative examples of concepts.

- Teaching the system under study was done by introducing the basic programming structures through analysis of blocks of CICS code and through analogy to COBOL.

- Teaching the failure modes was done by introducing the heuristic flowchart in interactive video. At its highest level, the cells in the flowchart are a taxonomy of program update types. This taxonomy was taught using conventional coordinate concept teaching strategies.

- Teaching the heuristics was done by presentation of each of the production systems ("rules of thumb") shown in the flowchart to be relevant to each type of program update. Practice was two level: first, in an on line CBT format, learners were asked to analyze a typical CICS maintenance request and select the production systems which applied to that problem. Then, learners were given a "final exercise" involving handling of a simulated maintenance request and manual rewriting of CICS code.

The course is structured to facilitate self paced mastery learn-

ing. However, practice is insufficient to lead to true automaticity for any of the skills taught.

While the course is a conscious attempt to apply some of the principles identified in this paper within the constraints of a practical, commercial development project, it is not without compromise. To complete the course within constraints of time and budget, a number of design decisions had to be made which are not fully supported by the research cited above. Thus, the project helped us identify a number of topics of interest to instructional designers which do not appear to be fully explored by research. These topics will be discussed in the next section.

WHAT WE DON'T KNOW ABOUT TEACHING TROUBLESHOOTING

The research cited above, and our experience with the CICS project, leads to articulation of a number of questions which are as yet not fully answered in the literature. These include:

1. How should knowledge be represented? In the CICS project, flowcharts were used to represent what was really a heuristic process. Concept hierarchies were used for the concept analysis. However, other authors have used a large variety of representation techniques, especially for production systems, heuristics, and rule structures. As yet there are no standard recommendations for the representation system to use in a cognitive task analysis. Scandura (1986) argues that the various representation systems have varying strengths, much as do different computer programming languages. If this is so, then what is needed are recommendations for when to use each representation system.

2. Should the strategy components of the skill be directly verbalized and taught, or should they be acquired inductively through practice? Chaiklin (1984) is typical of those who advocate direct verbal teaching of cognitive strategies, while Derry and Murphy (1986) are representative of those who seem to argue for inductive modeling (discovery) of strategies through practice, with little or no verbalization of the strategy.

3. Do the principles of teaching procedures apply to cognitive strategies and heuristics? Principles of procedure teaching such as those summarized by Salisbury, Richards and Klein (1985) may apply, but it may be that other techniques are more appropriate.

4. How should simulations be constructed for practicing the procedural knowledge components of troubleshooting skills? There are indications that carefully constructed but unrealistic simulations are more effective than realistic simulations, especially early in the learning process (see, for example, Munro, Fehling and Towne, 1985). However, empirically validated guidelines for constructing such simulations are only fragmentary.

5. How can a designer predict when the benefits of achieving automaticity outweigh the development costs for the extended practice sequences? Authors such as Elio (1986) argue that automaticity is needed for low level algorithmic components of problem solving, in order to manage attentional loading in higher level components. If this is so, what decision rules can a developer use to identify a priori the component subskills truly requiring automaticity?

Many more questions could be derived, of course. But it should be clear that research on the learning psychology side of the cognitive field far outstrips the research on the instructional psychology side. In many ways, the situation is analogous to that over twenty years ago, when behavioral theories did a much better job of describing learning than of prescribing instruction. At that time, a behaviorally based instructional psychology was developed. Perhaps it is time to do the same, using cognitive psychology as a basis.

REFERENCES

Andre, T. (1986) Problem Solving and Education. In Phye, G.D. and Andre, T., Cognitive Classroom Learning: Understanding, Thinking and Problem Solving. New York: Academic Press, 1986.

Barsalou, L.W. and Bauer, G.H. (1984) Discrimination nets as Psychological Models. Cognitive Science, 8, 1 26.

Brien, R. and Duchastel, P. (1986) Cognitive Task Analysis Underlying the Specification of Instructional Objectives. Programmed Learning and Educational Technology, 23, 4, November 1986, 363 370.

Chaiklin, S. (1984) On the Nature of Verbal Rules and Their Role in Problem Solving. Cognitive Science, 8, 131 155.

Cunningham, D.J. (1986) Good Guys and Bad Guys. Educational Communication and Technology Journal, 34, 1, 3 7.

Duncan, K.D. (1985) Representations of Fault Finding Problems and Development of Fault Finding Strategies. Programmed Learning & Educational Technology, 22, 2, 125 139.

Elio, R. (1986) Representation of Similar Well Learned Cognitive Procedures. Cognitive Science, 10, 41 75.

Elstein, A.S., Schulman, L.S. and Sprafka, S.A. (1978) Medical Problem Solving: An Analysis of Clinical Reasoning. Cambridge, Mass.: Harvard University Press.

Fredericksen, N. (1984) Implications of Cognitive Theory for Instruction in Problem Solving. Review of Educational Research, 54, 3, 363 407.

Gagné, R. M. (1954) An Analysis of Two Problem Solving Activities. Air Force Personnel & Training Research Center Research Bulletin AFPTRC TR 54 77.

Hewson, P.W. and Posner, G. J. (1984) The Use of Schema Theory in the Design of Instructional Materials: A Physics Example. Instructional Science, 13, 119 139.

Mager, R. (1982)Troubleshooting the Troubleshooting Course, or Debug d'Bugs. Belmont, CA: Pitman Learning.

Munro, A., Fehling, M. and Towne, D. (1985) Instruction Intrusiveness in Dynamic Simulation Training. Journal of Computer Based Instruction, 12, 2, 50 53.

Patel, V. and Groen, G. (1986) Knowledge Based Solution Strategies in Medical Reasoning. Cognitive Science, 10, 91 116.

Robbins, A. and Connors, M. (1987) CICS Maintenance. Course #5055 (Interactive Video and Video Assisted Instruction), Arlington Heights, IL: Advanced Systems, Inc.

Salisbury, D., Richards, B. and Klein, J. (1985) Designing Practice: A Review of Prescriptions from Instructional Design Theories. Journal of Instructional Development, 8, 4, 9 19.

Scandura, J.M. (1986) System Issues in Problem Solving Research. J. Structural Learning, 9, 1 13.

Schneider, W. and Shiffrin, R. (1977) Controlled and Automatic Human Information Processing: I. Detection, Search and Attention. Psychological Review, 84, 1, 1 66.

Tennyson, R. and Cocchiarella, M. (1986), An Emprically Based Instructicnal Design Theory for Teaching Concepts. Review of Educational Research, 56, 1, 40 71.